Assignment 1: Planning and Design

Harry Potter: The Designer's Stone

For the remainder of the semester, you will be working in teams on a relatively large software project. You will design and implement new functionality to add to an existing system that we will provide to you.

This assignment has been designed to be done in pairs, and we will not allow you to work on your own. You will be paired with another student in your lab by unit staff. If there is an odd number of students in your lab, your lab may have *one* team of three — the teaching staff at your campus will decide which team.

To help you manage your time on this project, we have divided it into three phases. These will be assessed separately. In this phase you will familiarise yourselves with the code base, decide on how to split up the tasks for the next assignment, and most importantly, create some preliminary designs for the extra functionality you will implement in the next phase (Assignment 2). You will extend the system again in Assignment 3, so do the best you can to keep your design and implementation for the system extensible and maintainable.

Managing and Submitting Your Work

This is a large project, and you will need a way to share files with your partner and unit staff. Instead of requiring you to submit your work on Moodle, we will provide you with a Monash-hosted GitLab repository that we can also read. There are a number of advantages to doing it this way:

- You learn to use Git to manage code and other software engineering artefacts. Git is a fully-featured modern version control system. It is the most widely used version control software for commercial, open-source, and hobbyist software projects today.
- It provides you with a mechanism for sharing files with your partner. You will be able to access your Monash GitLab repository from Monash and from home — anywhere you have internet access. Note that Git support is built into Eclipse, and almost all modern IDEs.
- You do not need to "submit" anything. Instead, ww will use the state of your Git repository at the due date and time. You just need to ensure that the master branch is ready for marking at that time.
- Your changes are automatically tracked. If you and your partner have a dispute about sharing the workload, unit staff can easily see whether you are adhering to your agreed tasks and timelines.
- Every time you make a change, you include a comment that summarises what was done. This greatly aids communication within your team.

Getting Started

The initial code base is available in a zip archive on Moodle. This archive includes a directory with some UML design documents that describe the code base.

Download the archive and create a project in your IDE so that you can explore it. You will need to spend considerable time trying out its functionality, and reading the code in the harrypotter.* packages.

Once you have your Monash GitLab repository, you will add this code base to it, and commit all changes to the repository.

Background

You will be working on a text-based "rogue-like" game. Rogue-like games are named after the first such program: a fantasy game named rogue. They were very popular in the days before home computers were capable of elaborate graphics, and still have a small but dedicated following. If you would like more information about roguelike games, a good site is http://www.roguebasin.com/.

The game is set in the Harry Potter universe, loosely based on J.K. Rowling's novels featuring the character Harry Potter. You are not required read the books — the specifications will give you enough information. If you have read the books or seen the movies, you are welcome to incorporate your knowledge of the story in your program.[1]

The main character is Harry Potter, a trainee wizard. Harry goes to a school for wizards called Hogwarts. Harry's best friends at Hogwarts are Hermione Granger and Ron Weasley. Harry and his friends have a series of adventures as Harry battles to defeat Voldemort, the leader of an evil group of wizards called the Death Eaters. In the Harry Potter world, ordinary (non-magical) humans are called Muggles. There are numerous magical creatures, such as dementors, dragons, unicorns, ghosts, house elves, basilisks and more.

As it stands, the game functions, but is missing a lot of desired functionality. Over the course of this project, you will incrementally add new features to the game.

Design Documents

For Assignment 1, you are not required to write any code. Instead, you must produce preliminary *design documentation* to explain how you are going to add the specified new functionality to the system. The new functionality is specified in the **Project Requirements** section.

We expect that you will produce *class diagrams* for all of the new features. Your class diagrams do not have to show the entire system. You only need to show the new parts, the parts that you expect to change, and enough information to let readers know where your new classes fit into the existing system. As it is likely that the precise names and signatures of methods will be refactored during development, you do not have to put them in this class diagram. However, the overall responsibilities of the class need to be documented *somewhere*, as you will need this information to be able to begin implementation.

To help us understand how your system will work, you must also write a *design rationale* to explain the choices you made. You must explain both how your proposed system will work and *why* you chose to do it that way.

You should use *interaction diagrams* (e.g. sequence or collaboration diagrams) to show how objects interact within your system. These are only needed for complex interactions. If you are unsure if an interaction is complex enough to require an explanatory diagram, consult your tutor.

The design (which includes *all* the diagrams and text that you create) must clearly show:

- what classes exist in your extended system.
- what the role and responsibilities of any new or significantly modified classes are.
- how these classes relate to and interact with the existing system.
- how the (existing and new) classes will interact to deliver the required functionality.

You are not required to create new documentation for components of the existing system that you *do not* plan to change.

Your design documents may be created with any tool that you choose — including a pen and paper if you can do so legibly. However, **you must create PDF, JPEG or PNG images of your design documents**

[1] If you want to learn more, see http://en.wikipedia.org/wiki/Harry_Potter

in your Git repository. We cannot guarantee that your marker will have the same tools available that you used (or the same versions).

If you want a UML diagramming tool, UMLet (http://www.umlet.com) is a free, easy-to-use UML diagramming tool that is particularly suitable for beginners. The diagrams in the initial code base were created using UMlet, and the source files are included.

As you work on your design, you must store it in your Git repository.

Git resources and policy

You are required to use Git to manage all project artefacts: code *and documents*. You have already starting using Git in your lab exercises. See the resources provided there and on Moodle for information on using Git. As in your lab tasks, a Git repository will be provided for you on the Monash GitLab server. **You must use the repository provided. Do not create a new repository**.

We will use your Git commit log to see who has been contributing to the project and who has not. That means that it is *very important* that you commit and push your work frequently — *including your design documentation*. It is also good practice to do this, as it makes it much less likely that you will lose work to a crash or accidentally deleting it.

Storing project artifacts elsewhere and commiting them to your repository just before the due date is **not** acceptable and will be penalized as not complying with the assignment specification. It is also highly risky — laptops get lost, hard disks become corrupt, and cloud services go offline at crucial moments (if the Monash GitLab server goes down, it is our problem, not yours).

Your team may adopt any policy you choose for branching and merging your repository; however, you will be marked on your master branch. If you have not integrated your changes to master by the due date and time, they will not be marked and you will receive zero marks for that functionality.

Work Breakdown Agreement

We require you to create a simple Work Breakdown Agreement (WBA) to let us know how you plan to divide the work between members of your team.

Your WBA must explain:

- who will be responsible for *producing* each deliverable (whether it is a part of the system, an internal document, or an externally-deliverable document),
- who will be responsible for *testing or reviewing* each deliverable, and
- the dates by which the deliverable, test, or review needs to be completed.

Both partners must accept this document. You can do this by adding your name to the WBA and pushing it to the server with a commit comment that says "I accept this Work Breakdown Agreement". (Rest assured that if the WBA is changed after this time, we will be able to tell!)

We do not require you to follow a template for the WBA. A simple .doc or .txt file that contains the information we need will be sufficient. This document is not worth marks in itself, but it is a hurdle requirement: your submission will not be accepted if there is no WBA. We will use it to make sure that you are allocating work fairly, and to make sure that we can hold individuals responsible for their contributions to the team.

We will use this document for accountability. If a team member complains that their partner is not contributing adequately to the project, or that their contributions are not being made in time to allow for review or debugging, FIT2099 staff will look at the partner's contribution in the logs. If we decide that the complaint is justified, this will be taken into account in the marking: students who do not contribute will be penalized and may fail, while students whose partner is not contributing adequately may have their mark scaled so as not to count any functionality that is missing due to the actions of their teammate.

Assignments 1 and 2 Requirements

You will be provided with a large codebase for a text-based game set in the Harry Potter universe. This game is based on an engine (located in the packages with prefix edu.monash.fit2099) that has been used for several years for games based in many different pop-culture universes. A simple set of packages that uses this engine for a game set in the Harry Potter universe has been created.

We have created a set of requirements that you must add to the Harry Potter system. If you are worried about the interpretation of these requirements, we encourage you to post a question on the unit Moodle Discussion Forum. Teaching staff will be quick to respond, and other students will also benefit from seeing the discussion. Always look on the forum before you post in case somebody has already posted a similar question!

Note that you are only expected to *design a solution* that can support these requirements for this initial Assignment. You will implement this solution (and refactor it if necessary) in Assignment 2. Assignment 3 will bring a new set of requirements for you to add to the system.

Leave Affordance

Some entities have been given a Take affordance (e.g. ring, sword, etc.). This allows an actor to pick the item up. There is currently no way for an actor to put down the item the actor is holding. This means that the actor cannot pick up anything else.

Add a Leave affordance that allows an actor to put down the item they are currently carrying. After this is done, the actor should be holding nothing, the item should be in the location of the actor when it was done, and the item should be able to be picked up again.

Give Affordance

Create a Give affordance that allows an actor holding an item to give that item to another actor that is in the same location. The actor to which the item is given should be prompted to either accept or reject the item as the very first part of its behaviour on its next turn. Non-player actors should accept the item with some probability – the exact acceptance behaviour is left up to you. If the item is rejected, it should stay with the actor that tried to give it.

Dementor

Dementors are evil creatures that feed on positive emotions. You need to implement a Dementor class (a subclass of HPActor), and add a few Dementors to the world. Dementors have the following behaviour:

- All Dementors are on team EVIL.
- A Dementor has a home base, which is its initial location.
- At the start of its turn (a turn is when its act() method is called), if a Dementor is in a location with one or more actors on a different team, it sucks energy out of the actors, reducing each actor's hitpoints by 40. It then does the movement behaviour described below.
- When a Dementor is at its home base, it waits for a random number of turns between 1 and 5 inclusive. It then randomly chooses a direction in which to travel, and randomly chooses how many steps to travel in that direction, up to a maximum of three steps. After that, on each turn it moves one step in its chosen direction until it cannot move further or it has moved the chosen number of steps. It then retraces its steps back to its base one step per turn.

Wand

This is a much more complicated task than the others. It is recommended that you only start this once you are sure that you have the other requirements working correctly. Implementing this task will require creating several classes, and adding functionality to existing classes.

A wand is a magical object that allows the actor holding it to cast spells. Create a subclass of HPEntity called Wand. To make the Wand class work, we will also need an action that allows the actor holding a

wand to cast a spell. You will need to decide how to make this action available to an actor holding a wand. An actor must be holding a wand in order to cast spells.

When an actor chooses to cast a spell, it is necessary to check which spells the actor knows, and get the actor to select the spell they want to cast. The actor also has to choose which other actor in the current location it wants to cast the spell on. When the actor has chosen a known spell and a target in the current location to cast it on, the action for the spell is executed.

Examples of spells include (you should try to implement 2 different types):

- *Expelliarmus*: causes the actor it is cast on to drop any item they are holding
- *Immobulus:* Freezes the actor it is cast upon
- *Sectumsempra:* Causes wounds as if by an invisible sword
- *Avada Kedavra*: kills the actor it is cast on instantly
- *Protego:* Protection against magical attacks

Potions

Health potions found in certain (possibly hidden) locations around the world can restore a certain or a random number of lost hit points.

Infrastructure

The engine packages are the way they are for good reasons. **Do not modify the engine code!** Doing this will make the game engine much less reusable — and cost marks.

Starting Coding

You are **not required to do any coding for Assignment 1**. You are free to start coding as soon as you like though. Indeed doing so may help you to understand the existing code base better, and the feasibility of your design ideas. No new or changed code in your repository will affect your mark for Assignment 1 (or even be looked at during marking).

Submission instructions

You must put your design documents and work breakdown agreement (in one of the acceptable file formats listed earlier) in the design-docs folder of your Monash GitLab repository.

The due date for this assignment is *Tuesday 22nd January at 23:59*. Note that this is later than it says in the unit guide. The reasons for this have been explained on Moodle. We will mark your Assignment 1 based on the state of the "master" branch of your Monash GitLab repository at that time.

Unless a team member has applied for and received special consideration according to the Monash Special Consideration Policy,[2] late submissions will be penalized at 10% per day late.

It is both team members' responsibility to ensure that the correct versions of the documentation are present in the repository by the due date and time. Once both teammates have agreed on a final Assignment 1 submission, do not make further commits to the master branch of the repository until the due date has passed, without the agreement of your teammate. If you want to continue to make changes to the repository for some reason, make another branch.

Remember:

Don't Repeat Yourself! (DRY)

Reduce Dependencies! (ReD)

[2] http://www.monash.edu/exams/changes/special-consideration

Getting Marking

You must explain your submitted documents to your tutor during your lab in the week of the deadline. Each team will be allocated a 15 minute slot in which to be interviewed. Both team members must explain the artefacts they have produced, and the rationale behind them.

The interview is a hurdle for getting m

Marking Criteria

This assignment will be marked on:

Design completeness Does your design support the functionality we have specified?

Design quality Does your design take into account the programming and design principles we have discussed in lectures? Look for the principles like

Practicality Can your design be implemented as it is described in your submission?

Following the brief Does your design comply with the constraints we have placed upon it — for instance, does your design leave the engine untouched, as required?

Documentation quality Does your design documentation clearly communicate your proposed changes to the system? This can include:

- UML notation consistency and appropriateness.
- consistency between artifacts.
- clarity of writing.
- level of detail (this should be sufficient but not overwhelming)